INSTITUTO FEDERAL Bahia Campus Vitória da Conquista	**LISTA DE EXERCÍCIOS 06**	
	CURSO: Bacharelado em Sistemas de Informação	**MODALIDADE:** Ensino Superior
	MÓDULO/SEMESTRE/SÉRIE: 3º	**PERÍODO LETIVO:** 2024.1
	DISCIPLINA: Linguagem de Programação II	**CLASSE:** 20241.3.119.1N
	DOCENTE: Alexandro dos Santos Silva	

INSTRUÇÕES

- Para resolução das questões abaixo, será admitido o uso apenas da sintaxe adotada para escrita de programas em Java.

1. Considere uma sequência de números inteiros armazenados em um arquivo de texto, estando em cada linha de texto apenas um número. Implemente uma classe utilitária, que disponha de método estático **`main`** no qual ocorra leitura de tais números e, após isso, computação da média dos mesmos. Quando da execução da aplicação, deverá ser fornecida localização e/ou nome do arquivo de texto.

2. Readapte a classe utilitária da questão anterior de tal modo que seja possível fornecer a quantidade de números inteiros a serem extraídos do arquivo de texto. Caso a quantidade de números armazenados no arquivo de texto seja inferior à quantidade informada, a computação da média deverá ser abortada; se a quantidade for, por sua vez, superior, considere a leitura apenas dos números armazenados nas primeiras linhas de textos (de acordo com a quantidade indicada), seguindo-se a isso computação da média.

3. Considere a existência de dois arquivos de texto nos quais estejam armazenados, respectivamente e linha a linha, primeiros nomes e últimos sobrenomes (a título de exemplo, arquivos semelhantes a estes estão disponíveis no endereço **https://drive.google.com/drive/folders/1cQgajBzeBzySxRetUAdxAllBmaTQkxR_**). Implemente uma classe utilitária, que disponha de método estático **`main`** na qual seja gerada uma lista de nomes completos de pessoas acompanhadas de suas respectivas idades, estando cada nome completo representado por um nome e por um sobrenome extraídos respectivamente de tais arquivos de texto. Em relação à quantidade de pessoas, deverá ser fornecido número inteiro indicativo desta quantidade, quando do início da execução da aplicação. A seleção dos nomes e sobrenomes com base nas relações contidas nos arquivos de texto deve ocorrer de forma aleatória, bem como a definição da idade de cada pessoa (esta última, entre 0 e 99).

   Observação: sugere-se que nomes e sobrenomes contidos nos arquivos de textos, após lidos ou extraídos, sejam armazenados em listas para posterior seleção aleatória, quando da constituição dos nomes completos de cada pessoa (neste caso, pode se considerar, inclusive, uso de implementações de listas disponibilizadas na biblioteca padrão da linguagem Java, a exemplo de **`java.util.ArrayList<E>`**). No tocante à seleção aleatória e à definição, igualmente, aleatória das idades, sugere-se o uso do método **`nextInt`** da classe **`java.util.Random`**; tal método retorna um número pseudorrandômico entre 0 (zero) e um valor inteiro indicado na forma de parâmetro para o método (uma aplicação do método segue-se abaixo).

```
01  Random gerador = new Random();                 // instanciação de gerador
02  for (int i = 0; i < 5; i++) {                  // geração de 5 números inteiros
03     System.out.println(gerador.nextInt(100));  // gerador de enésimo número (entre 0 e 99)
04  }
```

4. Readapte a classe utilitária da questão anterior de tal modo que a lista de pessoas geradas seja armazenada em um terceiro arquivo, havendo em cada linha apenas um nome completo de pessoa e sua respectiva idade separadas por um símbolo de vírgula. Quando da realização das operações de escrita no referido arquivo de texto, certifique-se de que os dados das pessoas sejam armazenados em ordem alfabética (por nome completo).

   Observação: para a ordenação das pessoas com base em seus respectivos nomes completos, considere qualquer algoritmo computacional. Demonstração, por exemplo, da **ordenação por inserção** segue-se abaixo utilizando-se do método **`compareTo`** da classe **`java.lang.String`**, o qual retorna um número inteiro que, se igual a 0 (zero), é indicativo de que a *string* passada como parâmetro é igual à *string* com a qual ela é comparada; por sua vez, se o número retornado é negativo, a *string* passada como parâmetro é anterior, em termos de ordem alfabética, à *string* com a qual é comparada, ao passo que, se positivo, a *string* passada como parâmetro é posterior, em termos de ordem alfabética, à *string* com a qual é comparada.

```
01  // inicialização de array com nomes (primeiro nome e último sobrenome)
02  String[] nomes = new String[] {"Francisco Silva",
03      "Alexandre Souza",
04      "Jennifer Carvalho",
05      "Alexandre Cardoso"};
06
07  // ordenação por inserção dos nomes
08  for (int i = 1; i < nomes.length; i++) {
09      String nomeTemp = nomes[i];         // enésimo nome
10
11      int j = i - 1;                      // índice de nome anterior ao enésimo nome em comparação
12
13      /* realocação de nomes anteriores para índices imediatamente subsequentes
14       * até que seja encontrado nome em ordem alfabética anterior ao enésimo nome */
15      while (j >= 0 && nomes[j].compareTo(nomeTemp) > 0) {
16      nomes[j + 1] = nomes[j];
```

```
17      j--;
18      }
19
20      /* atribuição de enésimo nome em índice de acordo com sua posição em relação aos
21      * nomes anteriores comparados */
22      nomes[j + 1] = nomeTemp;
23  }
24
25  for (int i = 0; i < nomes.length; i++)  // listagem de nomes
26      System.out.println(nomes[i]);
27  }
```

5. Readapte a classe utilitária da questão anterior de tal modo que os dados das pessoas sejam encapsulados em objetos da classe **`Pessoa`**, cuja implementação se segue abaixo. Observe que a classe é seriarizável, motivo pelo qual exige-se que objetos dela instanciados sejam gravados diretamente em arquivo (ao invés, portanto, do armazenamento dos dados de cada pessoa em uma única linha de texto, da forma como descrito na questão anterior).

```
01  import java.io.Serializable;
02
03  public class Pessoa implements Comparable<Pessoa>, Serializable {
04
05     private static final long serialVersionUID = 1L;
06
07     private String nome;
08     private String sobreNome;
09     private int idade;
10
11     public Pessoa(String nome, String sobreNome, int idade) {
12        this.nome = nome;
13        this.sobreNome = sobreNome;
14        this.idade = idade;
15     }
16
17     public String getNome() {
18        return nome;
19     }
20
21     public String getSobreNome() {
22        return sobreNome;
23     }
24
25     public int getIdade() {
26        return idade;
27     }
28
29     // retorno de nome completo (nome + sobrenome)
30     public String getNomeCompleto() {
31        return nome + " " + sobreNome;
32     }
33
34     // comparação com outro objeto com base nos respectivos nomes completos
35     public int compareTo(Pessoa outraPessoa) {
36        return getNomeCompleto().compareTo(outraPessoa.getNomeCompleto());
37     }
38
39  }
```

Observação: conforme demostrado na codificação acima, a classe **`Pessoa`** implementa o método **`compareTo`** da interface **`java.util.Comparable<T>`**, de modo a permitir comparação entre objetos daquela classe com base na ordem alfabética dos nomes completos (nome e sobrenome). Trata-se de método invocado indiretamente em caso de ordenação de objetos que estejam armazenados em um *array* genéricos da classe **`java.lang.Object`**, quando da execução do método estático **`sort`** da classe **`java.util.Arrays`**, indicando-se como parâmetro para este último exatamente aquele *array*.

6. Implemente uma classe utilitária, que disponha de método estático **`main`** na qual sejam lidos objetos serializados da classe gravados em arquivo gerado na questão anterior, seguindo-se a isso cálculo e exibição de idade média das pessoas representadas em tais objetos.